IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant	: James R. WASON	Group Art Unit: 2176
Appln. No.	: 10/606,547	Examiner: Maikhanh Nguyen
Filed	: June 26, 2003	Confirmation No.:5225
For	: RICH TEXT HANDLING FOR A WEB APPLICATION	

APPEAL BRIEF UNDER 37 C.F.R. §41.37

Commissioner for Patents
U.S. Patent and Trademark Office
Customer Service Window, Mail Stop Appeal Brief-Patents
Randolph Building
401 Dulany Street
Alexandria, VA 22314

Sir:

This appeal is from the Examiner's rejections of claims 1 – 15, 19 – 35 and 43 – 50 as set forth in the Final Office Action dated September 15, 2006. A Notice of Appeal and a Request for Pre-Appeal Brief Review with the associated fee under 37 C.F.R. §41.20(b)(1) were submitted on December 15, 2006. Payment of the requisite fee under 37 C.F.R. §41.20(b)(2) is submitted herewith. No additional fee is believed to be required for filing the instant Appeal Brief. However, if extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. §1.136(a), and any fees required therefor are hereby authorized to be charged to Deposit Account No. 09-0457.

(I) REAL PARTY IN INTEREST

The real party in interest is International Business Machines Corporation, assignee of the entire interest in the above-identified application by an assignment recorded in the U.S. Patent and Trademark Office on March 31, 2004 at Reel 014510 and Frame 0594.

(II) RELATED APPEALS AND INTERFERENCES

The Appellant, their legal representatives and the Assignee are not currently aware of any appeals, interferences, or judicial proceedings that may directly affect or be directly affected by or have some bearing on the Board's decision in this appeal. Attached hereto is a Related Proceedings Appendix showing no related appeals or interferences.

(III) STATUS OF THE CLAIMS

In the Final Office Action dated December 15, 2006 ("Final Office Action"), claims 1 - 50 are pending, of which claims 1 – 15, 19 – 35 and 43 – 50 are rejected, claims 16 – 18 are allowed, claims 36 – 38 are objected to, and claims 39 – 42 are withdrawn from consideration. Appellant submits that claims 1 – 15, 19 – 35 and 43 – 50, as presented in the Amendment filed July 5, 2006, are being appealed, and are listed in the "Claims Appendix" attached herewith.

(IV) STATUS OF THE AMENDMENTS

As discussed above, an Amendment under 37 C.F.R. §1.111 ("Amendment") was filed July 5, 2006, in which claims 1, 12, 16, and 24 were amended. The Amendment was entered at the time.

(V) SUMMARY OF THE CLAIMED SUBJECT MATTER

Independent Claim 1

By way of non-limiting example, the invention provides a method of representing and managing rich text for use by Web-based applications and browsers as implemented in a machine (see, e.g., page 2, lines 11 – 18, page 6, lines 12 – 20, and Figures 15A and 15B). The method comprises providing one or more classes for use by the applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text (see, e.g., page 6, lines 21 – 27). The method further comprises representing the rich text in the memory structure representation (see, e.g., page 7, lines 1 – 12, page 9, line 10 – page 14, line 23 and Figure 16). The method further comprises editing the rich text in a document using the memory structure representation to perform editing functions on the document having the rich text as managed and created by the one or more classes (see, e.g., page 16, line 25 – page 18, line 23, and Figures 10 and 11A – 11D).

Independent Claim 24

By way of non-limiting example, the invention provides a method of representing and managing documents having rich text for use by Web-based applications and browsers as implemented in a machine (see, e.g., page 2, lines 11 – 18, page 6, lines 12 – 20, and Figures 15A and 15B). The method comprises representing the rich text in the memory structure representation (see, e.g., page 7, lines 1 – 12, page 9, line 10 – page 14, line 23 and Figure 16). The method further comprises providing one or more classes for use by the applications to create the memory structure representation, the one or more classes including a rich text list class to create a rich text list node and to manage one or more rich text nodes and a rich text class to

create the one or more rich text nodes each representing a unit of the rich text (see, e.g., page 6, lines 21 – 27 and page 7, line 3 – page 8, line 8). The method further comprises providing well-formed segments of text to the one or more current rich text nodes from a rich text list node to initialize the current rich text nodes for representing rich text in a document (see, e.g., page 9, line 10 – page 14, line 23).

Independent Claim 43

By way of non-limiting example, the invention provides an apparatus for providing a means for representing and managing rich text for use by Web based applications and browsers (see, e.g., page 2, line 27 – page 3, line 4). The means includes a component representing rich text in a memory structure representation (see, e.g., page 7, lines 1 – 12, page 9, line 10 – page 14, line 23 and Figure 16). The means further includes a component providing one or more classes for use by the Web based applications and browsers to create the memory structure representation, wherein the one or more classes includes a) a rich text list class for managing one or more rich text nodes, and b) a rich text class to create one or more rich text nodes each representing a unit of rich text and its attributes (see, e.g., page 7, line 3 – page 8, line 8).

Independent Claim 48

By way of non-limiting example, the invention provides a computer program product comprising a computer usable medium having a computer readable program code embodied in the medium (see, e.g., page 3, lines 5 – 14). The computer program product comprises a first computer program code to provide one or more classes for use by Web based applications and browsers to at least create and manage one or more rich text nodes in a memory structure

representation representative of rich text (see, e.g., page 6, lines 21 – 27 and page 7, line 3 – page 8, line 8). The computer program product further comprises a second computer program code to represent the rich text in the memory structure representation (see, e.g., page 7, lines 1 – 12, page 9, line 10 – page 14, line 23 and Figure 16). The computer program product further comprises a third computer program code to edit rich text in a document using the memory structure representation to perform editing functions on the document having rich text as managed and created by the one or more classes (see, e.g., page 16, line 25 – page 18, line 23, and Figures 10 and 11A – 11D).

(VI) GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

(A) Claims 1 – 15, 19, 20, 24 – 31, 33 – 35, 43 – 45 and 47 – 50 are rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,480,206 B2 issued to Prinzing ("Prinzing '206") in view of U.S. Patent No. 6,470,364 B1 issued to Prinzing ("Prinzing '364").

(B) Claims 21 – 23, 32 and 46 are rejected under 35 U.S.C. §103(a) as being unpatentable over Prinzing '206 in view of Prinzing '364 and further in view of U.S. Patent No. 6,085,206 issued to Domini et al. ("Domini").

(VII) ARGUMENTS

(A) Claims 1 – 15, 19, 20, 24 – 31, 33 – 35, 43 – 45 and 47 – 50 rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,480,206 B2 issued to Prinzing ("Prinzing '206") in view of U.S. Patent No. 6,470,364 B1 issued to Prinzing ("Prinzing '364").

Claims 1 – 15, 19 and 20

The rejection of claims 1 – 15, 19 and 20 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

The Examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach all the claim limitations. MPEP § 2142. Appellants respectfully submit that the applied references do not teach or suggest all of the claim limitations.

The present invention generally relates to rich text capability for Web based applications and browsers, and more specifically, to a system and method for representing and controlling rich text in memory and various text representations. Independent claim 1 recites, in pertinent part:

> A method of representing and managing rich text for use by Web based applications and browsers as implemented in a machine, the method comprising the steps of:
>
> providing one or more classes for use by the applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text . . . .

The applied references do not teach or suggest this combination of features.

The Examiner asserts that Prinzing '206 teaches or suggests, at column 2, lines 31 – 61, column 3, lines 43 – 53 and the abstract, all of the features of claim 1, including "providing one or more classes for use by the applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text", except for the applications being "Web based applications or browsers". However, the Examiner asserts that Prinzing '364 teaches or suggests Web-based applications or browsers at column 4, line 46 – column 5, line 16 and column 11, lines 31 – 54, and that it would have been obvious to one of ordinary skill in the art to combine these features. Appellant respectfully disagrees.

Even assuming *arguendo* that it would have been obvious to combine the two Prinzing references, which Appellant does not concede, such a combination does not teach or suggest all of the features of the independent claim. For example, the combination of references does not teach or suggest a Web based application or browser, in combination with the remaining features of the respective independent claim.

Contrary to the Examiner's assertion, Prinzing '364 does not teach or suggest providing one or more classes for use by Web based applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text. Specifically, Prinzing '364 discloses a GUI editor application that generates a text component corresponding to a selectable user interface style (e.g., Windows user interface style, MacIntosh user interface

style, or Motif user interface style) and the type of content associated with the text component (e.g., Java, RTF, or HTML). More specifically, Prinzing '364 discloses at column 4, line 28 that:

> . . . an improved editor used on a computer system is provided that generates a text component corresponding to a selectable user interface style and the type of content associated with the text component. This improved design decouples the selection of the user interface style from the type of text a text component can edit. This allows a text component used to edit a particular type of text to be customized to many different user interface styles. Similarly, a text component designed for a particular user interface style can be customized to edit a number of different types of text.

Further, Prinzing '364 discloses the use of editor kits, which are dependent upon the type of content associated with the text component (e.g., Java editor kit, RTF editor kit, or HTML editor kit). Specifically, Prinzing '364 discloses at col. 4, line 46 that:

> Implementations consistent with the present invention facilitate generating text components in a GUI application based on text type. FIG. 2 is a block diagram illustrating the use of customized text editors in such a GUI application. This example includes editor kit objects 202, a text user interface object 203, a text component 204 being displayed on a display screen, and a model 206 with the text information used by a GUI application. Editor kit objects 202 include a Java editor kit 208, a rich text format (RTF) editor kit 210, a hypertext markup language (HTML) editor kit 212, and a text editor kit 214. The editor kits used to used to customized editors for editing Java source code, RTF text, HTML source code, and text source respectively.

Additionally, Prinzing '364 discloses at col. 11, line 36 that:

> Editor pane 516 determines if default Editor Kit 610 is capable of supporting the particular type of text (step 711). If the default Editor Kit does not support the text type, a new editor kit corresponding to the text type is instantiated (step 712). In practice, Text Type Registry 517 can be dynamically updated immediately before the text component is generated or can be updated statically by installing an Editor Kit before an application is executed. If Text Type registry 517 is dynamically updated, an application may include a new editor kit or provide the location of such an editor kit upon execution. For example, a new editor kit can be downloaded from a server on the Internet.

> The instance of Editor kit 610 is used to customize the editor within Editor Pane 516 (step 718). Editor pane 516 then combines the selected user interface style with the customized editor to display a customized text component in the GUI application (step 720).

As discussed below, these applications are not Web based applications or browsers, as they are merely GUI editor applications resident on a stand-alone computer in a non-Web based or non-browser environment.

Specifically, the cited portions of Prinzing '364 do not teach the use of Web based applications or browsers. Rather, Prinzing '364 discloses the different types of text that may be edited in the GUI application. By listing such types of text, including Java source code and HTML source code, though, Prinzing '364 does not teach the feature "providing one or more classes for use by the applications to at least create and manage one or more rich text nodes . . ." where the applications are "Web based applications and browsers." Rather, the cited portion of Prinzing '364 is silent as to whether such a GUI application using Java or other source code is on a stand-alone computer, or connected to a network, much less is a Web based application or a browser. Additionally, although Prinzing '364 does teach that "a new editor kit can be downloaded from a server on the Internet," this passage does not teach that the new editor kit itself is a Web based application or browser. In fact, Prinzing '364 teaches that the new editor kit can be "downloaded" and "installed", suggesting that the new editor kit is not, in fact, a Web based application or browser, but rather, is resident and functions on the user's computer, not through the Internet.

As should be understood by those of skill in the art, a Web based application is an application that is used in an online environment and a browser is a software application that enables a user to display and interact with text, images, and other information typically located

on a web page at a website on the World Wide Web or a local area network. This is in contrast to a non-Web based application, which is resident and functions on a user's stand-alone computer, as disclosed in Prinzing '364.

By way of explanation, a text editor is a software application used for editing plain text. An HTML editor is an editing interface software application for creating and editing HTML code. Moreover, the HTML editor is a basic text editor with extra functionality for the creation and manipulation of HTML code. The extra functionality provided by an HTML editor kit, provides templates, toolbars and keyboard shortcuts in the GUI application to allow a programmer to quickly insert common HTML elements and structures and compile an HTML file from within the editor.

Thus, Appellant submits an HTML editor is a GUI programming software application that allows a programmer to more easily write and edit HTML code. More specifically, instead of writing HTML code manually line by line, the HTML editor provides an interface allowing a programmer to more easily create and edit HTML code (e.g., allowing a programmer to cut and paste HTML code, quickly insert common HTML code elements, highlight source code syntax, etc.) using templates, toolbars and keyboard shortcuts.

Appellant submits that the text editor, and more specifically the HTML text editor, disclosed in Prinzing '364 is not a Web based application or a browser, as the Examiner asserts. Rather, Prinzing '364 discloses a GUI text editor application for editing, e.g., HTML programming language. While the output of the GUI text editor application disclosed in Prinzing '364 may ultimately be utilized by a Web based application or browser, the GUI text editor application itself is not a Web based application or a browser.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Claims 24 – 31 and 33 – 35

The rejection of claims 24 – 31 and 33 -35 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

The Examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach all the claim limitations. MPEP § 2142. Appellants respectfully submit that the applied references do not teach or suggest all of the claim limitations.

The present invention generally relates to rich text capability for Web based applications and browsers, and more specifically, to a system and method for representing and controlling rich text in memory and various text representations. Independent claim 24 recites, in pertinent part:

> A method of representing and managing documents having rich text for use by Web based applications and browsers as implemented in a machine, the method comprising the steps of:
>
> representing rich text in a memory structure representation;
>
> providing one or more classes for use by the applications to create the memory structure representation, the one or more classes including a rich text list class to create a rich text list node and to manage one or more

> rich text nodes and a rich text class to create the one or more rich text nodes each representing a unit of the rich text . . . .

The Examiner asserts that Prinzing '206 teaches or suggests, at column 3, lines 43 – 53, column 5, line 43 – 62 and the abstract, all of the features of claim 24, including "providing one or more classes for use by the applications to create the memory structure representation", except for the applications being "Web based applications or browsers". However, the Examiner asserts that Prinzing '364 teaches or suggests Web-based applications or browsers at column 4, line 46 – column 5, line 16 and column 11, lines 31 – 54, and that it would have been obvious to one of ordinary skill in the art to combine these features. Appellant respectfully disagrees.

Even assuming *arguendo* that it would have been obvious to combine the two Prinzing references, which Appellant does not concede, such a combination does not teach or suggest all of the features of the independent claim. For example, the combination of references does not teach or suggest a Web based application or browser, in combination with the remaining features of the respective independent claim.

Contrary to the Examiner's assertion, Prinzing '364 does not teach or suggest providing one or more classes for use by Web based applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text. Specifically, Prinzing '364 discloses a GUI editor application that generates a text component corresponding to a selectable user interface style (e.g., Windows user interface style, MacIntosh user interface style, or Motif user interface style) and the type of content associated with the text component (e.g., Java, RTF, or HTML). More specifically, Prinzing '364 discloses at column 4, line 28 that:

> . . . an improved editor used on a computer system is provided that generates a text component corresponding to a selectable user interface style and the type of content associated with the text component. This improved design decouples the selection of the user interface style from

> the type of text a text component can edit. This allows a text component used to edit a particular type of text to be customized to many different user interface styles. Similarly, a text component designed for a particular user interface style can be customized to edit a number of different types of text.

Further, Prinzing '364 discloses the use of editor kits, which are dependent upon the type of content associated with the text component (e.g., Java editor kit, RTF editor kit, or HTML editor kit). Specifically, Prinzing '364 discloses at col. 4, line 46 that:

> Implementations consistent with the present invention facilitate generating text components in a GUI application based on text type. FIG. 2 is a block diagram illustrating the use of customized text editors in such a GUI application. This example includes editor kit objects 202, a text user interface object 203, a text component 204 being displayed on a display screen, and a model 206 with the text information used by a GUI application. Editor kit objects 202 include a Java editor kit 208, a rich text format (RTF) editor kit 210, a hypertext markup language (HTML) editor kit 212, and a text editor kit 214. The editor kits used to used to customized editors for editing Java source code, RTF text, HTML source code, and text source respectively.

Additionally, Prinzing '364 discloses at col. 11, line 36 that:

> Editor pane 516 determines if default Editor Kit 610 is capable of supporting the particular type of text (step 711). If the default Editor Kit does not support the text type, a new editor kit corresponding to the text type is instantiated (step 712). In practice, Text Type Registry 517 can be dynamically updated immediately before the text component is generated or can be updated statically by installing an Editor Kit before an application is executed. If Text Type registry 517 is dynamically updated, an application may include a new editor kit or provide the location of such an editor kit upon execution. For example, a new editor kit can be downloaded from a server on the Internet.
>
> The instance of Editor kit 610 is used to customize the editor within Editor Pane 516 (step 718). Editor pane 516 then combines the selected user interface style with the customized editor to display a customized text component in the GUI application (step 720).

As discussed below, these applications are not Web based applications or browsers, as they are merely GUI editor applications resident on a stand-alone computer in a non-Web based or non-browser environment.

Specifically, the cited portions of Prinzing '364 do not teach the use of Web based applications or browsers. Rather, Prinzing '364 discloses the different types of text that may be edited in the GUI application. By listing such types of text, including Java source code and HTML source code, though, Prinzing '364 does not teach the feature "providing one or more classes for use by the applications to at least create and manage one or more rich text nodes . . ." where the applications are "Web based applications and browsers." Rather, the cited portion of Prinzing '364 is silent as to whether such a GUI application using Java or other source code is on a stand-alone computer, or connected to a network, much less is a Web based application or a browser. Additionally, although Prinzing '364 does teach that "a new editor kit can be downloaded from a server on the Internet," this passage does not teach that the new editor kit itself is a Web based application or browser. In fact, Prinzing '364 teaches that the new editor kit can be "downloaded" and "installed", suggesting that the new editor kit is not, in fact, a Web based application or browser, but rather, is resident and functions on the user's computer, not through the Internet.

As should be understood by those of skill in the art, a Web based application is an application that is used in an online environment and a browser is a software application that enables a user to display and interact with text, images, and other information typically located on a web page at a website on the World Wide Web or a local area network. This is in contrast to a non-Web based application, which is resident and functions on a user's stand-alone computer, as disclosed in Prinzing '364.

By way of explanation, a text editor is a software application used for editing plain text. An HTML editor is an editing interface software application for creating and editing HTML code. Moreover, the HTML editor is a basic text editor with extra functionality for the creation and manipulation of HTML code. The extra functionality provided by an HTML editor kit, provides templates, toolbars and keyboard shortcuts in the GUI application to allow a programmer to quickly insert common HTML elements and structures and compile an HTML file from within the editor.

Thus, Appellant submits an HTML editor is a GUI programming software application that allows a programmer to more easily write and edit HTML code. More specifically, instead of writing HTML code manually line by line, the HTML editor provides an interface allowing a programmer to more easily create and edit HTML code (e.g., allowing a programmer to cut and paste HTML code, quickly insert common HTML code elements, highlight source code syntax, etc.) using templates, toolbars and keyboard shortcuts.

Appellant submits that the text editor, and more specifically the HTML text editor, disclosed in Prinzing '364 is not a Web based application or a browser, as the Examiner asserts. Rather, Prinzing '364 discloses a GUI text editor application for editing, e.g., HTML programming language. While the output of the GUI text editor application disclosed in Prinzing '364 may ultimately be utilized by a Web based application or browser, the GUI text editor application itself is not a Web based application or a browser.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Claims 43 – 45 and 47

The rejection of claims 43 – 45 and 47 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

The Examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach all the claim limitations. MPEP § 2142. Appellants respectfully submit that the applied references do not teach or suggest all of the claim limitations.

The present invention generally relates to rich text capability for Web based applications and browsers, and more specifically, to a system and method for representing and controlling rich text in memory and various text representations. Independent claim 43 recites, in pertinent part:

> An apparatus for providing a means for representing and managing rich text for use by Web based applications and browsers, the apparatus comprising:
> a component representing rich text in a memory structure representation;
> a component providing one or more classes for use by the Web based applications and browsers to create the memory structure representation, wherein the one or more classes includes . . . .

The applied references do not teach or suggest this combination of features.

The Examiner asserts that Prinzing '206 teaches or suggests, at column 3, lines 43 – 53, column 5, line 43 – 62 and the abstract, all of the features of claim 43, including "a component providing one or more classes for use by the . . . applications . . . to create the memory structure representation . . .", except for the applications being "Web based applications or browsers". However, the Examiner asserts that Prinzing '364 teaches or suggests Web-based applications or browsers at column 4, line 46 – column 5, line 16 and column 11, lines 21 – 54, and that it would have been obvious to one of ordinary skill in the art to combine these features. Appellant respectfully disagrees.

Even assuming *arguendo* that it would have been obvious to combine the two Prinzing references, which Appellant does not concede, such a combination does not teach or suggest all of the features of the independent claim. For example, the combination of references does not teach or suggest a Web based application or browser, in combination with the remaining features of the respective independent claim.

Contrary to the Examiner's assertion, Prinzing '364 does not teach or suggest providing one or more classes for use by Web based applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text. Specifically, Prinzing '364 discloses a GUI editor application that generates a text component corresponding to a selectable user interface style (e.g., Windows user interface style, MacIntosh user interface style, or Motif user interface style) and the type of content associated with the text component (e.g., Java, RTF, or HTML). More specifically, Prinzing '364 discloses at column 4, line 28 that:

> . . . an improved editor used on a computer system is provided that generates a text component corresponding to a selectable user interface style and the type of content associated with the text component. This improved design decouples the selection of the user interface style from the type of text a text component can edit. This allows a text component used to edit a particular type of text to be customized to many different

> user interface styles. Similarly, a text component designed for a particular user interface style can be customized to edit a number of different types of text.

Further, Prinzing '364 discloses the use of editor kits, which are dependent upon the type of content associated with the text component (e.g., Java editor kit, RTF editor kit, or HTML editor kit). Specifically, Prinzing '364 discloses at col. 4, line 46 that:

> Implementations consistent with the present invention facilitate generating text components in a GUI application based on text type. FIG. 2 is a block diagram illustrating the use of customized text editors in such a GUI application. This example includes editor kit objects 202, a text user interface object 203, a text component 204 being displayed on a display screen, and a model 206 with the text information used by a GUI application. Editor kit objects 202 include a Java editor kit 208, a rich text format (RTF) editor kit 210, a hypertext markup language (HTML) editor kit 212, and a text editor kit 214. The editor kits used to used to customized editors for editing Java source code, RTF text, HTML source code, and text source respectively.

Additionally, Prinzing '364 discloses at col. 11, line 36 that:

> Editor pane 516 determines if default Editor Kit 610 is capable of supporting the particular type of text (step 711). If the default Editor Kit does not support the text type, a new editor kit corresponding to the text type is instantiated (step 712). In practice, Text Type Registry 517 can be dynamically updated immediately before the text component is generated or can be updated statically by installing an Editor Kit before an application is executed. If Text Type registry 517 is dynamically updated, an application may include a new editor kit or provide the location of such an editor kit upon execution. For example, a new editor kit can be downloaded from a server on the Internet.
>
> The instance of Editor kit 610 is used to customize the editor within Editor Pane 516 (step 718). Editor pane 516 then combines the selected user interface style with the customized editor to display a customized text component in the GUI application (step 720).

As discussed below, these applications are not Web based applications or browsers, as they are merely GUI editor applications resident on a stand-alone computer in a non-Web based or non-browser environment.

Specifically, the cited portions of Prinzing '364 do not teach the use of Web based applications or browsers. Rather, Prinzing '364 discloses the different types of text that may be edited in the GUI application. By listing such types of text, including Java source code and HTML source code, though, Prinzing '364 does not teach the feature "providing one or more classes for use by the applications to at least create and manage one or more rich text nodes . . ." where the applications are "Web based applications and browsers." Rather, the cited portion of Prinzing '364 is silent as to whether such a GUI application using Java or other source code is on a stand-alone computer, or connected to a network, much less is a Web based application or a browser. Additionally, although Prinzing '364 does teach that "a new editor kit can be downloaded from a server on the Internet," this passage does not teach that the new editor kit itself is a Web based application or browser. In fact, Prinzing '364 teaches that the new editor kit can be "downloaded" and "installed", suggesting that the new editor kit is not, in fact, a Web based application or browser, but rather, is resident and functions on the user's computer, not through the Internet.

As should be understood by those of skill in the art, a Web based application is an application that is used in an online environment and a browser is a software application that enables a user to display and interact with text, images, and other information typically located on a web page at a website on the World Wide Web or a local area network. This is in contrast to a non-Web based application, which is resident and functions on a user's stand-alone computer, as disclosed in Prinzing '364.

By way of explanation, a text editor is a software application used for editing plain text. An HTML editor is an editing interface software application for creating and editing HTML code. Moreover, the HTML editor is a basic text editor with extra functionality for the creation and manipulation of HTML code. The extra functionality provided by an HTML editor kit, provides templates, toolbars and keyboard shortcuts in the GUI application to allow a programmer to quickly insert common HTML elements and structures and compile an HTML file from within the editor.

Thus, Appellant submits an HTML editor is a GUI programming software application that allows a programmer to more easily write and edit HTML code. More specifically, instead of writing HTML code manually line by line, the HTML editor provides an interface allowing a programmer to more easily create and edit HTML code (e.g., allowing a programmer to cut and paste HTML code, quickly insert common HTML code elements, highlight source code syntax, etc.) using templates, toolbars and keyboard shortcuts.

Appellant submits that the text editor, and more specifically the HTML text editor, disclosed in Prinzing '364 is not a Web based application or a browser, as the Examiner asserts. Rather, Prinzing '364 discloses a GUI text editor application for editing, e.g., HTML programming language. While the output of the GUI text editor application disclosed in Prinzing '364 may ultimately be utilized by a Web based application or browser, the GUI text editor application itself is not a Web based application or a browser.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Claims 48 – 50

The rejection of claims 48 – 50 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

The Examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach all the claim limitations. MPEP § 2142. Appellants respectfully submit that the applied references do not teach or suggest all of the claim limitations.

The present invention generally relates to rich text capability for Web based applications and browsers, and more specifically, to a system and method for representing and controlling rich text in memory and various text representations. Independent claim 48 recites, in pertinent part:

> . . . a first computer program code to provide one or more classes for use by Web based applications and browsers to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text;
>
> a second computer program code to represent the rich text in the memory structure representation; and
>
> a third computer program code to edit rich text in a document using the memory structure representation to perform editing functions on the document having rich text as managed and created by the one or more classes.

The applied references do not teach or suggest this combination of features.

The Examiner asserts that Prinzing '206 teaches or suggests, at column 2, lines 44 – 61, column 3, lines 43 – 53 and the abstract, all of the features of claim 48, including "a first computer program code to provide one or more classes for use by . . . applications . . . to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text", except for the applications being "Web based applications or browsers". However, the Examiner asserts that Prinzing '364 teaches or suggests Web-based applications or browsers at column 4, line 46 – column 5, line 16 and column 11, lines 31 – 54, and that it would have been obvious to one of ordinary skill in the art to combine these features. Appellant respectfully disagrees.

Even assuming *arguendo* that it would have been obvious to combine the two Prinzing references, which Appellant does not concede, such a combination does not teach or suggest all of the features of the independent claim. For example, the combination of references does not teach or suggest a Web based application or browser, in combination with the remaining features of the respective independent claim.

Contrary to the Examiner's assertion, Prinzing '364 does not teach or suggest providing one or more classes for use by Web based applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text. Specifically, Prinzing '364 discloses a GUI editor application that generates a text component corresponding to a selectable user interface style (e.g., Windows user interface style, MacIntosh user interface style, or Motif user interface style) and the type of content associated with the text component (e.g., Java, RTF, or HTML). More specifically, Prinzing '364 discloses at column 4, line 28 that:

> . . . an improved editor used on a computer system is provided that generates a text component corresponding to a selectable user interface

> style and the type of content associated with the text component. This improved design decouples the selection of the user interface style from the type of text a text component can edit. This allows a text component used to edit a particular type of text to be customized to many different user interface styles. Similarly, a text component designed for a particular user interface style can be customized to edit a number of different types of text.

Further, Prinzing '364 discloses the use of editor kits, which are dependent upon the type of content associated with the text component (e.g., Java editor kit, RTF editor kit, or HTML editor kit). Specifically, Prinzing '364 discloses at col. 4, line 46 that:

> Implementations consistent with the present invention facilitate generating text components in a GUI application based on text type. FIG. 2 is a block diagram illustrating the use of customized text editors in such a GUI application. This example includes editor kit objects 202, a text user interface object 203, a text component 204 being displayed on a display screen, and a model 206 with the text information used by a GUI application. Editor kit objects 202 include a Java editor kit 208, a rich text format (RTF) editor kit 210, a hypertext markup language (HTML) editor kit 212, and a text editor kit 214. The editor kits used to used to customized editors for editing Java source code, RTF text, HTML source code, and text source respectively.

Additionally, Prinzing '364 discloses at col. 11, line 36 that:

> Editor pane 516 determines if default Editor Kit 610 is capable of supporting the particular type of text (step 711). If the default Editor Kit does not support the text type, a new editor kit corresponding to the text type is instantiated (step 712). In practice, Text Type Registry 517 can be dynamically updated immediately before the text component is generated or can be updated statically by installing an Editor Kit before an application is executed. If Text Type registry 517 is dynamically updated, an application may include a new editor kit or provide the location of such an editor kit upon execution. For example, a new editor kit can be downloaded from a server on the Internet.
>
> The instance of Editor kit 610 is used to customize the editor within Editor Pane 516 (step 718). Editor pane 516 then combines the selected user interface style with the customized editor to display a customized text component in the GUI application (step 720).

As discussed below, these applications are not Web based applications or browsers, as they are merely GUI editor applications resident on a stand-alone computer in a non-Web based or non-browser environment.

Specifically, the cited portions of Prinzing '364 do not teach the use of Web based applications or browsers. Rather, Prinzing '364 discloses the different types of text that may be edited in the GUI application. By listing such types of text, including Java source code and HTML source code, though, Prinzing '364 does not teach the feature "providing one or more classes for use by the applications to at least create and manage one or more rich text nodes . . ." where the applications are "Web based applications and browsers." Rather, the cited portion of Prinzing '364 is silent as to whether such a GUI application using Java or other source code is on a stand-alone computer, or connected to a network, much less is a Web based application or a browser. Additionally, although Prinzing '364 does teach that "a new editor kit can be downloaded from a server on the Internet," this passage does not teach that the new editor kit itself is a Web based application or browser. In fact, Prinzing '364 teaches that the new editor kit can be "downloaded" and "installed", suggesting that the new editor kit is not, in fact, a Web based application or browser, but rather, is resident and functions on the user's computer, not through the Internet.

As should be understood by those of skill in the art, a Web based application is an application that is used in an online environment and a browser is a software application that enables a user to display and interact with text, images, and other information typically located on a web page at a website on the World Wide Web or a local area network. This is in contrast to a non-Web based application, which is resident and functions on a user's stand-alone computer, as disclosed in Prinzing '364.

By way of explanation, a text editor is a software application used for editing plain text. An HTML editor is an editing interface software application for creating and editing HTML code. Moreover, the HTML editor is a basic text editor with extra functionality for the creation and manipulation of HTML code. The extra functionality provided by an HTML editor kit, provides templates, toolbars and keyboard shortcuts in the GUI application to allow a programmer to quickly insert common HTML elements and structures and compile an HTML file from within the editor.

Thus, Appellant submits an HTML editor is a GUI programming software application that allows a programmer to more easily write and edit HTML code. More specifically, instead of writing HTML code manually line by line, the HTML editor provides an interface allowing a programmer to more easily create and edit HTML code (e.g., allowing a programmer to cut and paste HTML code, quickly insert common HTML code elements, highlight source code syntax, etc.) using templates, toolbars and keyboard shortcuts.

Appellant submits that the text editor, and more specifically the HTML text editor, disclosed in Prinzing '364 is not a Web based application or a browser, as the Examiner asserts. Rather, Prinzing '364 discloses a GUI text editor application for editing, e.g., HTML programming language. While the output of the GUI text editor application disclosed in Prinzing '364 may ultimately be utilized by a Web based application or browser, the GUI text editor application itself is not a Web based application or a browser.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

(B) Claims 21 – 23, 32 and 46 rejected under 35 U.S.C. §103(a) as being unpatentable over Prinzing '206 in view of Prinzing '364 and further in view of U.S. Patent No. 6,085,206 issued to Domini et al. ("Domini").

Claims 21 – 23

The rejection of claims 21 – 23 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

Claims 21 – 23 depend from allowable claim 1. Claim 21 recites, in pertinent part:

> responding to a spell checking request;
> presenting a spell check panel that displays spelling alternatives to a misspelled word associated with the one or more rich text nodes; and
> accepting a spelling substitution.

Appellant submits that no proper combination of the Prinzing '206, Prinzing '364 and Domini teaches these features.

As discussed above with respect to claim 1, the applied references do not teach or suggest providing one or more classes for use by the application to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text, where the application is a Web based application or browser. As the Examiner cited Domini merely for a teaching of features of dependent claims 21- 23 , 32 and 46, Appellant submits Domini does not teach or suggest providing one or more classes for use by the application to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text, where the application is a Web based application or browser. Thus, Domini does not cure the above-noted deficiencies of the rejection of independent claim 1.

For these reasons, Appellant submits that any proper combination of Prinzing '206, Prinzing '364 and Domini would not teach all the features of claims 21 – 23.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Claim 32

The rejection of claim 32 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

Claim 32 depends from allowable claim 24. Claim 32 recites, in pertinent part:

> . . . wherein the providing one or more classes step further comprises the step of: providing a spell checker class for use by the applications for locating replacement words in the document having rich text.

Appellant submits that no proper combination of the Prinzing '206, Prinzing '364 and Domini teaches these features.

As discussed above with respect to claim 24, the applied references do not teach or suggest providing one or more classes for use by the applications to create the memory structure representation, where the applications are Web based applications or browsers. As the Examiner cited Domini merely for a teaching of features of dependent claims 21- 23 , 32 and 46, Appellant submits Domini does not teach or suggest providing one or more classes for use by the applications to create the memory structure representation, where the applications are Web based applications or browsers. Thus, Domini does not cure the above-noted deficiencies of the rejection of independent claim 24.

For these reasons, Appellant submits that any proper combination of Prinzing '206, Prinzing '364 and Domini would not teach all the features of claims 32.

Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Claim 46

The rejection of claim 46 under 35 U.S.C. §103(a) is in error, the decision of the Examiner to reject these claims should be reversed, and the application should be remanded to the Examiner.

Claims 46 depends from allowable claim 43. Claim 46 recites, in pertinent part:

> . . . further comprising a component for providing spell checking using the memory structure representation.

Appellant submits that no proper combination of the Prinzing '206, Prinzing '364 and Domini teaches these features.

As discussed above with respect to claim 43, the applied references do not teach or suggest a component providing one or more classes for use by Web based applications or browsers to create the memory structure representation. As the Examiner cited Domini merely for a teaching of features of dependent claims 21- 23 , 32 and 46, Appellant submits Domini does not teach or suggest a component providing one or more classes for use by Web based applications or browsers to create the memory structure representation. Thus, Domini does not cure the above-noted deficiencies of the rejection of independent claim 43.

For these reasons, Appellant submits that any proper combination of Prinzing '206, Prinzing '364 and Domini would not teach all the features of claim 46.

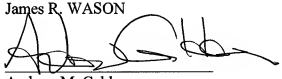
Therefore, the rejection is improper because the applied references do not teach or suggest each and every feature of the claimed invention.

Conclusion

In view of the foregoing remarks, Appellant submits that claims 1 – 15, 19 – 35 and 43 – 50 are patentably distinct from the prior art of record and are in condition for allowance. Accordingly, Appellant respectfully requests that the Board reverse the Examiner's rejection of claims 1 – 15, 19 – 35 and 43 – 50, and remand the application to the Examiner for withdrawal of the above-noted rejections.

Respectfully submitted,

James R. WASON



Andrew M. Calderon
Reg. No. 38,093

GREENBLUM & BERNSTEIN, P.L.C.
1950 Roland Clarke Place
Reston, VA 20191
(703) 716-1191

(VIII) CLAIMS APPENDIX

Upon entry of the amendment filed on July 5, 2006, the following is a listing of the claims involved in this appeal.

1. (Previously Presented) A method of representing and managing rich text for use by Web based applications and browsers as implemented in a machine, the method comprising the steps of:

providing one or more classes for use by the applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text;

representing the rich text in the memory structure representation; and

editing the rich text in a document using the memory structure representation to perform editing functions on the document having the rich text as managed and created by the one or more classes.

2. (Original) The method of claim 1, wherein the providing the one or more classes includes the steps of:

providing a rich text list class for managing the one or more rich text nodes in the memory structure representation;

providing a rich text class to create the one or more rich text nodes each representing a unit of rich text and its attributes; and

instantiating the rich text list class and the rich text class.

3. (Original) The method of claim 1, wherein the representing rich text step includes representing string representations.

4. (Original) The method of claim 3, wherein the string representations comprise at least one of a character large object (CLOB), hyper-text markup language (HTML), extensible markup language (XML), plain text, and spell check text.

5. (Original) The method of claim 1, wherein the providing one or more classes step includes providing rich text attributes, wherein the attributes include at least one of font face, font size, font color, italicized, underlined, and bold.

6. (Original) The method of claim 1, wherein the providing one or more classes step includes providing properties associated with the one or more rich text nodes, the properties comprising at least one of a line break, a table, an image, a link, and text.

7. (Original) The method of claim 1, wherein the rich text node comprises a table node for defining a table.

8. (Original) The method of claim 7, wherein the table node includes at least one of a table header node and a table body node, for defining the characteristics and format of the table.

9. (Original) The method of claim 8, wherein the table header comprises one or more heading cell nodes, each heading cell node defining another rich text node.

10. (Original) The method of claim 8, wherein the table body node comprises one or more table row nodes for defining an individual row within the table.

11. (Original) The method of claim 10, wherein the one or more table row nodes comprise one or more row cell nodes for defining rich text in a cell in the individual row, each of the one or more row cell nodes defining another rich text node.

12. (Previously Presented) The method of claim 1, further comprising the steps of:

providing well-formed segments of text to a current rich text node of the one or more rich text nodes from a rich text list node;

parsing the well-formed segments of text;

assigning unparsed segments of text to the current rich text node's text attribute; and

resolving the current rich text node's text attribute by extracting tag in formation and setting attributes in the current rich text node, the attributes including at least one of font face, font size, font color, italicized, underlined, and bold.

13. (Original) The method of claim 12, wherein the providing well-formed segments step comprises the steps of:

suppressing certain tags associated with some the unparsed segments by changing starting and ending tags to substitution strings;

checking whether the starting and ending tags are in proper order and eliminating pairs of the starting and the ending tags that have null content;

converting some of the substitution strings to original values; and

reconstituting the well-formed segments of text into one string when pairs of starting and end tags are eliminated.

14. (Original) The method of claim 12, wherein the providing well-formed segments step comprises the steps of:

restoring table related tags; and

breaking the well-formed segments at table tags and organizing the broken segments into a new rich text list node with entries of at least one of vectors and string.

15. (Original) The method of claim 12, wherein the text is at least one of hypertext mark-up language (html) and extensible mark-up language (xml).

16. (Previously Presented) A method of representing and managing rich text for use by applications as implemented in a machine, the method comprising the steps of:

providing one or more classes for use by the applications to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text;

representing the rich text in the memory structure representation; and

editing the rich text in a document using the memory structure representation to perform editing functions on the document having the rich text as managed and created by the one or more classes, further comprising the steps of:

providing well-formed segments of text to a current rich text node of the one or more rich text nodes from a rich text list node;

parsing the well-formed segments of text;

assigning unparsed segments of text to the current rich text node's text attribute; and

resolving the current rich text node's text attribute by extracting tag in formation and setting attributes in the current rich text node, the attributes including at least one of font face, font size, font color, italicized, underlined, and bold;

wherein the resolving step comprises the steps of:

a) reading the text attribute up to a first tag;

b) if the reading step produces a non-null string, then cloning the current rich text node to make a preceding rich text node and assigning to it all text before the tag;

c) checking whether the first tag has a matching end tag;

d) if there is a matching end tag, cloning the current rich text node to make a following rich text node and assigning to it any text after the matching end tag, then removing the text after the matching end tag;

e) resolving the information between the first tag and matching end tag to set up attributes in the current rich text node; and

f) repeating steps a) through e) until a null string is produced in step b).

17. (Original) The method of claim 16, further comprising the step of repeating steps a) through f) on one of the preceding rich text node and the following rich text node.

18. (Original) The method of claim 16, further comprising the step of when the first tag is one of an image tag and a link tag in step a), cloning the current rich text node to make the following rich text node and assigning to the following node the text after the first tag, then continuing with step c).

19. (Original) The method of claim 1, further comprising the steps of:

responding to a request for editing a document containing the rich text;

presenting rich text editing controls for editing the document; and

accepting changes to the document using one or more classes including a rich text class and a rich text list class for editing the document.

20. (Original) The method of claim 19, wherein the accepting changes step includes accepting changes to at least one of a table, a link, an image, and text.

21. (Original) The method of claim 19, wherein the responding step further comprises steps of:

responding to a spell checking request;

presenting a spell check panel that displays spelling alternatives to a misspelled word associated with the one or more rich text nodes; and

accepting a spelling substitution.

22. (Original) The method of claim 21, wherein the responding to a spell checking request step includes searching a spelling dictionary to locate one or more words for presentation in the spell check panel.

23. (Original) The method of claim 22, wherein the one or more words in the dictionary each have one or more associated signatures to aid in locating a match for the misspelled word.

24. (Previously Presented) A method of representing and managing documents having rich text for use by Web based applications and browsers as implemented in a machine, the method comprising the steps of:

representing rich text in a memory structure representation;

providing one or more classes for use by the applications to create the memory structure representation, the one or more classes including a rich text list class to create a rich text list node and to manage one or more rich text nodes and a rich text class to create the one or more rich text nodes each representing a unit of the rich text; and

providing well-formed segments of text to the one or more current rich text nodes from a rich text list node to initialize the current rich text nodes for representing rich text in a document.

25. (Original) The method of claim 24, further comprising the steps of:

instantiating the rich text list class and the rich text class; and

editing the rich text in the document using the rich text nodes created by the rich text class.

26. (Original) The method of claim 24, wherein the representing rich text step includes representing string representations, the string representations including at least one of a compressed format, hyper-text markup language (HTML), extensible markup language (XML), plain text, and spell check text.

27. (Original) The method of claim 24, wherein the rich text includes attributes of at least one of font face, font size, font color, italicized, underlined, and bold.

28. (Original) The method of claim 24 wherein the one or more rich text nodes includes properties, the properties comprising at least one of a line break, a table, an image, a link, and text.

29. (Original) The method of claim 24, wherein the one or more rich text node comprises a table node for defining a table and the table node includes at least one of a table header node and a table body node, for defining the characteristics and format of the table.

30. (Original) The method of claim 29, wherein the table header node comprises one or more heading cell nodes, each heading cell node defining another rich text node, and wherein the table body node comprises one or more table row nodes for defining an individual row within the table.

31. (Original) The method of claim 30, wherein the one or more table row nodes comprise one or more row cell nodes for defining rich text in a cell in the individual row, each of the one or more row cell nodes defining another rich text node.

32. (Original) The method of claim 24, wherein the providing one or more classes step further comprises the step of: providing a spell checker class for use by the applications for locating replacement words in the document having rich text.

33. (Original) The method of claim 24, wherein the providing well-formed segments step comprises the steps of:

converting some substitution strings to original values;

suppressing certain tags by changing starting and ending tags to substitution strings;

checking whether start and end tags are in proper order and eliminating pairs of start and end tags that have null content; and

reconstituting segments of text into one string when pairs of starting and end tags are eliminated.

34. (Original) The method of claim 24, wherein the providing well-formed segments step comprises the steps of: restoring table related tags; and breaking some of the unparsed segments at table tags and organizing the broken segments into a new rich text list node with entries of at least one of vectors and string.

35. (Original) The method of claim 24, wherein the providing well-formed segments of text step further comprising the steps of:

parsing the well-formed segments of text;

assigning unparsed segments of text to the current rich text node's text attribute; and

resolving the current rich text node's text attribute by extracting tag information and sets attributes in the current rich text node, the attributes including at least one of font face, font size, font color, italicized, underlined, and bold.

36. (Previously Presented) A method of representing and managing documents having rich text for use in a machine, the method comprising the steps of:

representing rich text in a memory structure representation;

providing one or more classes for use by the applications to create the memory structure representation, the one or more classes including a rich text list class to create a rich text list node and to manage one or more rich text nodes and a rich text class to create the one or more rich text nodes each representing a unit of the rich text; and

providing well-formed segments of text to the one or more current rich text nodes from a rich text list node to initialize the current rich text nodes for representing rich text in a document,

wherein the providing well-formed segments of text step further comprising the steps of:

parsing the well-formed segments of text;

assigning unparsed segments of text to the current rich text node's text attribute; and

resolving the current rich text node's text attribute by extracting tag information and sets attributes in the current rich text node, the attributes including at least one of font face, font size, font color, italicized, underlined, and bold,

wherein the resolving step comprises the steps of:

a) reading the text attribute up to a first tag;

b) if the reading step produces a non-null string, then cloning the current rich text node to make a preceding rich text node and assigning to it all text before the tag;

c) checking whether the first tag has a matching end tag;

d) if there is a matching end tag, cloning the current rich text node to make a following rich text node and assigning to it any text after the matching end tag, then removing the text after the matching end tag;

e) resolving the information between the first tag and matching end tag to set up attributes in the current node; and

f) repeating steps a) through e) until all a null string is produced in step b).

37. (Original) The method of claim 36, further comprising the step of repeating steps a) through f) on one of the preceding rich text node and the following rich text node.

38. (Original) The method of claim 36, further comprising the step of when the first tag is one of an image tag and a link tag in step a), cloning the current rich text node to make the following rich text node and assigning to the following node the text after the first tag, then continuing with step e).

39. (Withdrawn) A method of providing a spellchecker function for use with documents having rich text, the method comprising the steps of:

initializing a dictionary containing words;

creating at least one signature for each dictionary word;

keying the at least one signature to the dictionary word; determining that a word is misspelled by checking the dictionary for the misspelled word resulting in a null value;

creating at least one signature associated with the misspelled word; searching the dictionary using the at least one signature associated with the misspelled word and dictionary word to locate at least one replacement word with the same at least one signature; and

providing the at least one replacement word in the document having rich text.

40. (Withdrawn) The method of claim 39, wherein the at least one signature associated with the misspelled word and for each dictionary word is provided by extracting one or more letters and combining the one or more letters.

41. (Withdrawn) The method of claim 40, wherein the extracting one or more letters and combining step is provided according to at least one of the following:

a) when the dictionary word or misspelled word is less than three characters, the at least one signature is the dictionary word or misspelled word itself,

b) when the length of each of the dictionary word or misspelled word is greater than eight characters, one signature is the first half of the word,

c) when the length of the dictionary word or misspelled word is eight the first three and last three characters are each signatures,

d) when the length of the dictionary word or misspelled word is between four and seven, the first two characters and last two characters are each signatures,

e) when the length of the dictionary word or misspelled word equals four, the first two characters plus the last character is the signature,

f) when the length of the dictionary word or misspelled word is greater than four, the first four and the last four characters are each signatures, and

g) when the length of the dictionary word or misspelled word equals four, the first character plus the last two characters is a signature.

42. (Withdrawn) The method of claim 39, wherein the providing step includes providing more than one replacement words in an ordered list for selection, wherein the more than one replacement words are ordered based upon a score.

43. (Original) An apparatus for providing a means for representing and managing rich text for use by Web based applications and browsers, the apparatus comprising:

a component representing rich text in a memory structure representation;

a component providing one or more classes for use by the Web based applications and browsers to create the memory structure representation, wherein the one or more classes includes,

a) a rich text list class for managing one or more rich text nodes and

b) a rich text class to create one or more rich text nodes each representing a unit of rich text and its attributes.

44. (Original) The apparatus of claim 43, further comprising:

a component instantiating the rich text list class and the rich text class; and

a component editing rich text in a document using the rich text class.

45. (Original) The apparatus of claim 43, wherein the component for representing rich text includes representing a string, the string including at least one of a character large object (CLOB), hyper-text markup language (HTML), extensible markup language (XML), plain text, and spell check text.

46. (Original) The apparatus of claim 43, further comprising a component for providing spell checking using the memory structure representation.

47. (Original) The apparatus of claim 43, wherein the component for representing rich text in a memory structure representation and the component for providing one or more classes for use by the Web based applications and browsers is contained on at least one of a compact disc, a network, a library, a hard drive, a floppy disc, and a memory device.

48. (Previously Presented) A computer program product comprising a computer usable medium having a computer readable program code embodied in the medium, the computer program product includes:

a first computer program code to provide one or more classes for use by Web based applications and browsers to at least create and manage one or more rich text nodes in a memory structure representation representative of rich text;

a second computer program code to represent the rich text in the memory structure representation; and

a third computer program code to edit rich text in a document using the memory structure representation to perform editing functions on the document having rich text as managed and created by the one or more classes.

49. (Original) The computer program product of claim 48, wherein the computer program product further includes:

a fourth computer program code to provide a rich text list class for creating rich text list nodes and for managing the one or more rich text nodes in the memory structure representation;

a fifth computer program code to provide a rich text class to create the one or more rich text nodes each representing a unit of rich text and its attributes; and

a sixth computer program code to instantiate the rich text list class and the rich text class.

50. (Original) The computer program product of claim 49, wherein the computer program product further includes:

a seventh computer program code to provide well-formed segments of text to a current rich text node from a rich text list node;

an eighth computer program code to parse the well-formed segments of text;

a ninth computer program code to assign unparsed segments of text to the current rich text node's text attribute; and

a tenth computer program code to resolve the current rich text node's text attribute by extracting tag information and to set attributes in the current rich text node.

(IX) EVIDENCE APPENDIX

This section lists evidence submitted pursuant to 37 C.F.R. §§ 1.130, 1.131, or 1.132, or any other evidence entered by the Examiner and relied upon by Appellant in this appeal, and provides for each piece of evidence a brief statement setting forth where in the record that evidence was entered by the Examiner. Copies of each piece of Evidence are provided as required by 37 C.F.R. §41.37(c)(1)(ix).

NO.	EVIDENCE	BRIEF STATEMENT SETTING FORTH WHERE IN THE RECORD THE EVIDENCE WAS ENTERED BY THE EXAMINER
1	N/A	N/A

(X) RELATED PROCEEDINGS APPENDIX

Pursuant to 37 C.F.R. § 41.37(c)(1)(x) copies of the following decisions rendered by a court or the Board in any proceeding identified above in the Related Appeals and Interferences section.

NO.	TYPE OF PROCEEDING	REFERENCE NO.	DATE
1	N/A	N/A	N/A